HŪDAT Security Systems

Project Report

Speaker Verification System in a Security Application

Brian Bash
Thomas Jonell
Dustin Williams

Faculty Advisor:
Dr. Les Thede

Date:
04-30-04

Executive Summary 1
Problem Identification 2
Research 2
Biometric Decision 2
Table 1: Decision Matrix for Biometric Types 3
Decision Matrix Terms 4
Voice Biometrics 5
Constraints 7
System Design 8
Figure 1: Block Diagram of Speaker Verification System 8
The System Components 8
Programming Language 9
MATLAB Files 9
File Splitting and Formants 12
File Splitting 12
Formants 12
Lock Circuit 13
Figure 2: Lock Circuit Schematic 13
Testing 14
Cost Analysis 15
Table 2: Estimated Development Costs 15
Table 3: Estimated Product Costs 15
Gantt Chart 16
Fall Quarter 16
Winter Quarter 16
Spring Quarter 16
Figure 3: Project Gantt Chart 17
Conclusion 18
References 19
Appendix A – M-file Code 20
deriv.m 20
formant.m 20
lock.m 21
logon.m 22
newsplit.m 25
newtest.m 26
pitch.m 27
recwav.m 28
Appendix B – Sample User Interfaces 29
Example 1: Successful Login Attempt 29
Example 2: Three Failed Login Attempts 29
Example 3: Invalid User Name 30
Appendix C – MATLAB Plots 31

Executive Summary

A speaker verification system in a security application is developed. It involves the use of voice biometrics to distinguish between authorized users. The system deactivates an electric lock upon a successful match. The issue of cost is addressed.

Problem Identification

In today's society, we are in the middle of the "Information Age." Over the past twenty years, technology has made every attempt to try and keep up with the overwhelming demand. In the process, recent technology has spread into every facet of our lives. After September 11, 2001, society has looked considerably more to technology for security, something which was lacking before and is now fundamental.

A customer has a desire to control access to a secure room by means other than a traditional keycard or passcode. The idea is that these security measures can be compromised or stolen. After studying different approaches, the answer was determined to be biometrics.

Research

Biometrics are automated methods of identifying speech (recognition). They can also be used as a means of verifying the identification of the person (verification). When discussing the field, recognition and verification are often confused. Recognition deals with the understanding of speech content. A system may draw from a large database of known words and will determine what was said. Verification does not care what is said. It makes an identification to verify that the person is who he or she is claiming to be.

Biometrics is quickly becoming a way to ensure that secure items do not become compromised. This process is completed by comparing newly acquired data against data already in the verification system. Data is in the format of either physiological or behavioral characteristics. Physiological examples include facial recognition, finger prints, and retinal scans. Behavioral characteristics include, but are not limited to, signature verification and speaker verification. In this project, the use of speaker verification will be implemented.

Biometric Decision

Considering the multiple routes that the field of biometrics could take in a security application, **Table 1** explains the benefits (and drawbacks) of each type. After reviewing the results, it can be seen that speaker verification is a viable option considering its common and inexpensive hardware, its low cost, and its ease of use. In addition, it is felt that this would be the easiest system to implement. Its drawbacks in the areas of error and reliability can be negated somewhat by utilizing additional, more traditional verification procedures such as entering a passcode into a keypad.

Table 1: Decision Matrix for Biometric Types

Criteria	Fingerprint	Facial Recognition	Hand Geometry	Speaker Verification	Iris Scan
Verify	yes	yes	yes	yes	yes
ID	yes	no	no	no	yes
Accuracy	4	3	3	2	4
Reliability	3	2	2	1	3
Error Rate	1 in 500+	no data	1 in 500	1 in 50	1 in 131,000
Errors	dryness, dirt, age	lighting, age, glasses, hair	hand injury, age	noise, weather, colds	poor lighting
False Pos.	Ext. Diff.	Difficult	Very Diff.	Medium	Very Diff.
False Neg.	Ext. Diff.	Easy	Medium	Easy	Very Diff.
Security Level	3	2	2	2	3
Long-term Stability	3	2	2	2	3
User Acceptance	2	2	2	3	2
Intrusive	Somewhat	Non	Non	Non	Non
Ease of Use	3	2	3	3	2
Low Cost	yes	yes	no	yes	Non
Hardware	Special, cheap	Common, cheap	Special, mid-price	Common, cheap	Special, expensive
Standards	yes	?	?	?	?

Criteria	Retinal Scan	Signature Recognition	Keystroke Recognition	DNA
Verify	yes	yes	yes	yes
ID	yes	no	no	yes
Accuracy	4	2	1	4
Reliability	3	1	1	3
Error Rate	1 in 10,000,000	1 in 50	no data	no data
Errors	glasses	changing signatures	hand injury, tiredness	none
False Pos.	Ext. Diff.	Medium	Difficult	Ext. Diff.
False Neg.	Ext. Diff.	Easy	Easy	Ext. Diff.
Security Level	3	2	2	3
Long-term Stability	3	2	1	3
User Acceptance	2	2	3	1
Intrusive	Very	Non	Non	Extremely
Ease of Use	1	3	3	1
Low Cost	no	yes	yes	no
Hardware	Special, expensive	Special, mid-price	Common, cheap	Special, expensive
Standards	?	?	?	yes

This table can be found at: http://ct.ncsc.dni.us/biomet%20web/BMCompare.html [**1**]

Decision Matrix Terms

Verify - Whether or not the Biometric is capable of verification. Verification is the process where an input is compared to specific data previously recorded from the user to see if the person is who they claim to be.

ID - Whether or not the Biometric is capable of identification. Identification is the process where an input is compared to a large data set previously recorded from many people to see which person the user is.

Accuracy (1-4) - How well the Biometric is able to tell individuals apart. This is partially determined by the amount of information gathered as well as the number of possible different data results.

Reliability (1-3) - How dependable the Biometric is for recognition purposes.

Error Rate - This is calculated as the crossing point when graphed of false positives and false negatives created using this Biometric.

Errors - Typical causes of errors for this Biometric.

False Pos. - How easy it is to create a false positive reading with this biometric (someone is able to impersonate someone else).

False Neg. - How easy it is to create a false negative reading with this biometric (someone is able to avoid identification as oneself).

Security Level (1-3) - The highest level of security that this Biometric is capable of.

Long-term Stability (1-3) - How well this Biometric continues to work without data updates over long periods of time.

User Acceptance (1-3) - How willing the public is to use this Biometric.

Intrusiveness - How much the Biometric is considered to invade one's privacy or require interaction by the user.

Ease of Use (1-3) - How easy this Biometric is for both the user and the personnel involved.

Low Cost - Whether or not there is a low-cost option for this Biometric to be used.

Hardware - Type and cost of hardware required to use this Biometric.

Standards - Whether or not standards exist for this Biometric.

Voice Biometrics

Voice biometrics is the use of a person's voice as an identifying characteristic of the person. One area where people often become confused is the distinction between speech recognition and speaker verification. Put simply, speech recognition is "a speech-processing technology that recognizes what a person is saying" [**2**]. This is often applied in computer software intended for the use of the handicapped. You can probably recall programs that allow you to "speak" to your computer and it will act as a stenographer by taking down what you say in a word processor. Speaker verification, however, is the use of a person's voice to identify him or her, usually in a secure setting. Unlike speech recognition, it does not know what the speaker is saying. Instead, it utilizes a voiceprint database, extracts the reference voiceprint after an identity claim is made, gets a sample of speech from the user, converts this to a voiceprint and compares the two voiceprints through means of digital signal processing (DSP), utilizes some threshold of error to account for slight variations (possibly due to background noise, speaker illness, etc.), and accepts the claim or not [**2**].

There are four main types of speaker verification that are used today. These are text-dependent, text-dependent with speech recognition, text-prompted, and text-independent. Text-dependent involves something like an account number being initially typed into the system. Then a prompt occurs and a password is uttered. As for text-dependent verification with speech recognition, there may be an account number said. The system would verify the user as well as determine what was said. Text-prompted verification can involve an entered account number with the system then prompting the user to repeat pre-determined phrases. Finally, there is text-independent verification which is the hardest form to use. It is also, however, the most unobtrusive. Basically, it could involve a call to a bank. A system asks the user what they wish to do. The user then states an instruction (i.e. "I want to transfer $10,000 to my offshore account in the Cayman Islands."). As the system is using the processing the instruction (speech recognition), it is also using speaker verification to confirm the identity of the user [**2**]. All in all, examples of speaker verification are everywhere.

In the realm of security, speaker verification is utilized in applications such as data networks like BMC Software (password reset over the telephone using virtual help desk), Illinois Dept. of Revenue (off-site access to secure data networks), and INTRUST Bank (internal wire transfers). As for physical/site access, examples are used by the U.S. Immigration and Naturalization Service (entry to U.S. and Canada during off hours; port of entry at Scobey, Mont.), Girl Tech (door access control system and locked box for children), and the City of Baltimore (evening and weekend access to the five main city buildings).

Speaker verification is also used in fraud prevention in telephone network security (toll fraud) applications such as University of Maryland's College Park (toll-free long-distance lines for faculty and staff) and GTE TSI (integration of speaker verification into wireless security packages offered to carriers). In transaction security, there is the Home Shopping Network

(automated product-ordering over the telephone) and Glenview State Bank (transfer of money between accounts of a bank customer).

It is also used in monitoring. This is done in time and attendance monitoring such as SOC Credit Union and the Salvation Army. One very important application in monitoring can be seen in corrections monitoring. Speaker verification is used at the New York City Dept. of Probation (tracking of juvenile and adult probationers) and the Dane County Jail in Madison, Wisc. (monitoring of home-incarcerated offenders) [**2**].

Constraints

Upon looking at the entire project as a whole, certain issues needed to be brought up to determine its plausibility. Below are the constraints that were considered along with the project's impact upon them.

Economic
Non-standard parts in the hardware could be an issue for the consumer in maintenance, so it has been decided that standard parts will be used whenever possible. Also, software development packages could greatly increase the cost of development concerning licensure yet reduce the amount of time necessary for developing a prototype. The number of these packages used in the project will be reduced as much as reasonably possible. This will be done through research and decision matrices. Another issue to consider is the existence of similar systems. An attempt will be made to develop an original solution at a competitive price.

Environmental
It was determined that no environmental impact can be foreseen. Any issues that arise will be dealt with accordingly.

Manufacturability
Again, the issue of standard hardware comes up. The use of standard hardware will allow for ease of production and construction.

Sustainability
It is felt that proper documentation, sufficiently commented code, and a user's manual will need to be provided to allow for any necessary changes to be made in the future.

Health & Safety
Safety would be compromised if the system did not work properly, so all feasible scenarios will be tested to debug the system and ensure operation. A closed beta test is being considered to accomplish this. Also, a manual override (i.e. a key) will be implemented to circumvent any system failures that would jeopardize human life.

Social
Since this system will not be used by general society, no great social impact can be foreseen. Any issues that arise will be dealt with accordingly.

Polticial/Ethical
The collecting of data to identify individuals may pose a concern in the long run. Future applications of the technology could lead to abuse of collected bio-data. Issues of privacy and personal liberties may arise. The general scope of this project does not provide any political or ethical concerns.

System Design

The HUDAT Security System incorporates the use of a computer-driven microphone, a directory of reference audio samples, an audio processor, an interface, and a locking mechanism. An individual repeats the pass-phrase ("Please let me in.") into a microphone. From this, an audio sample is generated. The audio processor then compares the sample to the directory of samples and grants access by unlocking the door lock only if the new sample matches a previous authorized sample. A tolerance is incorporated to allow for slight changes in tonal quality, background noise and other factors to be determined. A block diagram of the system can be seen in **Figure 1**.

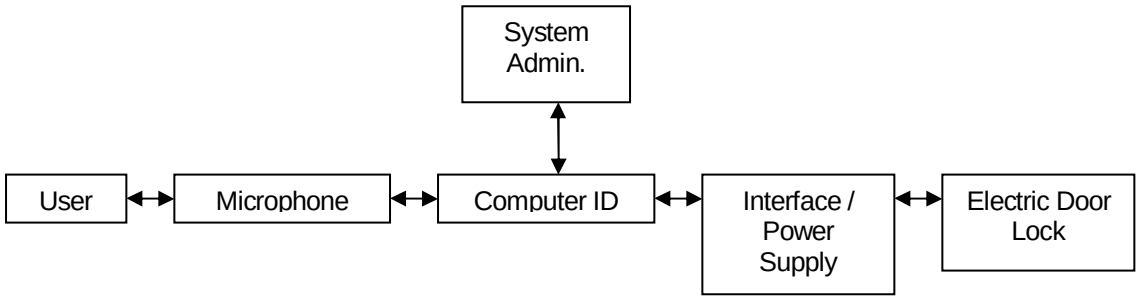


Figure 1: Block Diagram of Speaker Verification System

The System Components

User - A person wishing access into the secure area.

Microphone - A device capable of capturing an audio sample and relaying it to a computer.

System Administrator - A supervisor of the recognition system responsible for general maintenance of the system as well as maintaining the database. They are also responsible for manual identification if something would go wrong with the system (i.e. false negative due to an illness, etc.).

Computer ID - A device capable of processing the captured samples and interfacing with the database for comparison. Upon a successful match, it will send a signal to the interface.

Interface / Power Supply - A device that uses the computer signal to trigger a relay. This relay closes the circuit required to power the electric door lock.

Electric Door Lock - An electric door lock whose locking mechanism is normally locked and is unlocked upon receiving an electric signal of a specified voltage.

Programming Language

For the project, a programming language was needed to construct the code that would operate the system. Among the available languages, it was decided that the program MATLAB would be used to initially construct and debug the system. This was decided because MATLAB is a very diverse program written in C-code. It provides an intuitive interface, language, and a number of math and graphics functions. It also has many special functions that would be useful in development (i.e. *fft* for performing a Discrete Fourier Transform of a given signal) plus a wide array of other Digital Signal Processing (DSP) applications. It is the team's belief that the code written in MatLab (along with establishing a few functions) could then be put into C# code to develop an executable.

MATLAB Files

myfft.m [Created: 11-04-03]
The file was originally created to show the Fast Fourier Transform (FFT) of voice data that was sent through an FIR bandpass filter based around the range of the human voice (500-2000Hz) to eliminate unwanted noise. After filtering they were sent through a MatLab Hamming window function followed by an appropriately set up FFT function. After listening to the results of the filter, it was decided that the bandpass produced an unwanted muted sound. Instead a FIR lowpass filter was used, and the results were acceptable. The file was later modified to filter and perform the FFT on two given sets of voice data. The *myfft* function was used until the formant method replaced it.

peakfirst.m [Created: 1-5-04]
The *peakfirst* function was the first in a series of functions used to find the peaks in a set of data. The function *peakfirst* worked by first finding the peak entry in the data set and then progressively move forward through the data finding new peaks up to a specified index. This idea was flawed because if the first peak occurred halfway or near the end of the specified index, peaks preceding the first max wouldn't be found. Another flaw was if there were two peaks that were relatively large any peaks that occurred between them wouldn't be found. Because of these flaws, the *peakfirst* function was replaced by the *peak* function.

split.m [Created: 1-7-04]
The first splitting function *split* analyzed the voice data to determine when there was a "voiced" section and an "unvoiced" section, by "zeroing" any portions of the data that did not go above a certain threshold. The program then made a second pass through the new partially "zeroed" data and determined whether portions were long enough to be considered "voiced" and whether they were separated enough to be considered separate or the same spoken portion. The function worked as intended, but was the slowest part of the system, sometimes taking 30 seconds to split a file. The extra time needed came from the decision process in which the function decided whether a portion of the file was long enough to be considered "voiced" and had enough separation from the next "voiced" section. A change

was made in the collection of the voice samples by having the speaker clearly separate their words. With this change it was clear that a new splitting function was needed, because there was now no reason for the extra time taken in performing the split. The *newsplit* function replaced the *split* function by performing the split almost 10 times faster.

peak.m [Created: 1-13-04]
The *peak* function replaced the failed *peakfirst* function. This function attacked peak finding by a new course of action. Instead of searching forward from the first peak found, the next, and so forth, when the first peak was found, it and a window around it would be set to zero, removing it from being detected as a peak again. The next peak was then searched for, and it, like the first, was "zeroed". This was done until all the peaks above a certain threshold were found. When the switch from the FFT method to the formant method was made, peak was modified into the *peakf* function, with little success. Because of the way the formant data arranged, the "zeroing" method did not work, finding false peaks. The peak functions were replaced by an entirely different method with the *derv* function.

test.m [Created: 1-13-04]
The *test* function originally worked by taking the two sets of indexes given by the *peak* function and comparing them together. They were compared at first by how well the two indexes when paired as x-y coordinates lined up to a slope of 1. If they failed to line up to the slope of one, the test failed. It was then realized that the size of the two FFTs of the voiced samples were not of the same size, therefore the indexes would technically be wrong. In fact the indexes themselves shouldn't have been used at all because the indexes of an FFT represent a frequency. So a new function called *frec* was created to translate the FFT indexes to their proper frequencies. These new frequency sets were then sent to the test function to compare once more as x-y coordinates to a slope of 1. When the decision for using formants was made, this testing method was scrapped in favor of a new one comparing the difference of the sets to each other in the *newtest* function.

fullprog.m [Created: 1-17-04]
The program *fullprog* incorporates all of the FFT method functions. The *fullprog* program always worked as intended, but its individual functions never produced the desired results, so it was scrapped, being replaced by *final*.

formant.m [Created: 2-11-04]
The *formant* function takes in a voiced file and applies the Yule-Walker spectral power density MatLab function *pyulear* to it. The data is then aligned properly and sent to the *deriv* function to find peaks (formants), which index and value are returned along with the power-density curve. The *formant* function has worked as intended since it was created.

newsplit.m [Created: 2-16-04]
The *newsplit* function worked by utilizing the autocorrelation function *xcorr* in MatLab. The *newsplit* used the *xcorr* function via the *pitch* function. The *pitch* function applied the *xcoor* function in small overlapping windows on the original voice data, finding the maximum number of this autocorrelation and placing it in a new data set. This new data set was passed to the *newsplit* function. The new dataset represented an essentially amplified

version of the original voice data, making it incredibly easy to detect a “voiced” and “unvoiced” section. The new data set index was mapped to the original data set and the file was split apart by use of a magnitude threshold. Initial problems in mapping the data caused the *newsplit* function to cutoff the beginning and ending of a “voiced” section. This was solved by increasing the size of the “voiced” window after identification of the “voiced” section. Although this eliminated the cutoff problem, a new problem emerged involving the creation of extraneous “voiced” sections. This problem was solved by increasing the length threshold of a “voiced” section. With the above solutions, the *newsplit* function works as intended.

deriv.m [Created: 2-17-04]
The *deriv* function was created out of the need for peak finder that would not find false peaks. This function worked by utilizing numerical derivative roots. A peak always occurs in a function when its derivative transitions between positive and negative. The function incorporated this idea and finds peaks without problems.

newtest.m [Created: 2-24-04]
The function *newtest* compares the functions by squaring the difference of the formant peak indices together and then summing those squares. The smaller the sum, the more similar the two voices are. There have been no problems with this function.

final.m [Created: 2-25-04]
This program incorporates all the formant method functions into a single program.
The program currently works as intended.

lock.m [Created: 3-15-04]
This program unlocks the door strike. It communicates with the serial port of the computer on which it is running. The serial port is sent a signal for approximately 5 seconds, during which time the door lock “buzzes” due to AC power being supplied.

logon.m [Created: 3-15-04]
This program replaces *final*. It provides the user with a text-based interface to the entire system. Its structure works in the following manner:

1. Welcomes the user to the HUDAT Security System.
2. Prompts the user to enter their user name.
3. Verifies that the user name is allowed access to the secure area.
4. When ready, a key is pressed to initiate the 6-second recording time.
5. Upon completing recording, the system splits the new wave file using *newsplit*.
6. The system verifies that the recording was successful and the split was performed properly.
7. Upon split verification, the wave file associated with the user name is also split.
8. The formant function is used to test the four split portions of each wave file.
9. A number based upon the similarity of each file compared is returned by the formant function. A failure is indicated by -1, while two files with similar power spectral densities returns a positive integer (0 indicates an exact match).
10. A weighted grade is assigned to the return of each formant:

a. 0-25: Assigned a ‘2’
b. 26-250: Assigned a ‘1’
c. 251-500: Assigned a ‘0’
d. 501+ or -1 Assigned a ‘-1’

11. The grades from each of the four comparisons are totaled. A value of zero or greater is required for the system to generate a ‘pass’ response.
12. Upon passing, the *lock* function is called. Upon failing, the system reattempts the recording process for up to 3 trials. After 3 trials, the system resets.

File Splitting and Formants

File Splitting

In order to properly analyze a spoken file, the voiced sections need to be identified. Due to the dynamic range of the voice when speaking, determining when someone starts/stops speaking and when there is white noise can be difficult. The solution to enhancing the voiced sections of the voice sample was solved by use of autocorrelation. This is done by the autocorrelation of many small windows along the voice sample and then storing the max value of each window in a reference vector. The mean of the reference vector is then used as a threshold value to determine when a voiced section begins and ends. The indices of the starting and stopping points of the reference vector are taken and then mapped to the original voice sample in order to successfully split it. Appendix C shows a graphical comparison of an original voice sample and its reference vector.

Formants

After a file is split into individual words, each word needs to be analyzed to find out what makes it unique. The uniqueness of each word is measured by formants. A formant is a characteristic resonant region (peak) in the power spectral density (PSD) of a sound. The PSD is essentially the concentration of power at specific frequencies. The PSD is unique to each voice due to the resonant qualities of the vocal chords and the mouth/nasal cavities. The PSD is obtained by utilizing the Yule-Walker AR method to calculate a smooth PSD curve. The formants of the PSD curve are found at the peaks using a numerical derivative. Appendix C shows an example PSD curve with the formants marked.

Lock Circuit

This portion of the system involves the circuit between the program's voltage signal (via RS232 Serial) and the electric door lock. **Figure 2** shows the schematic of this circuit.

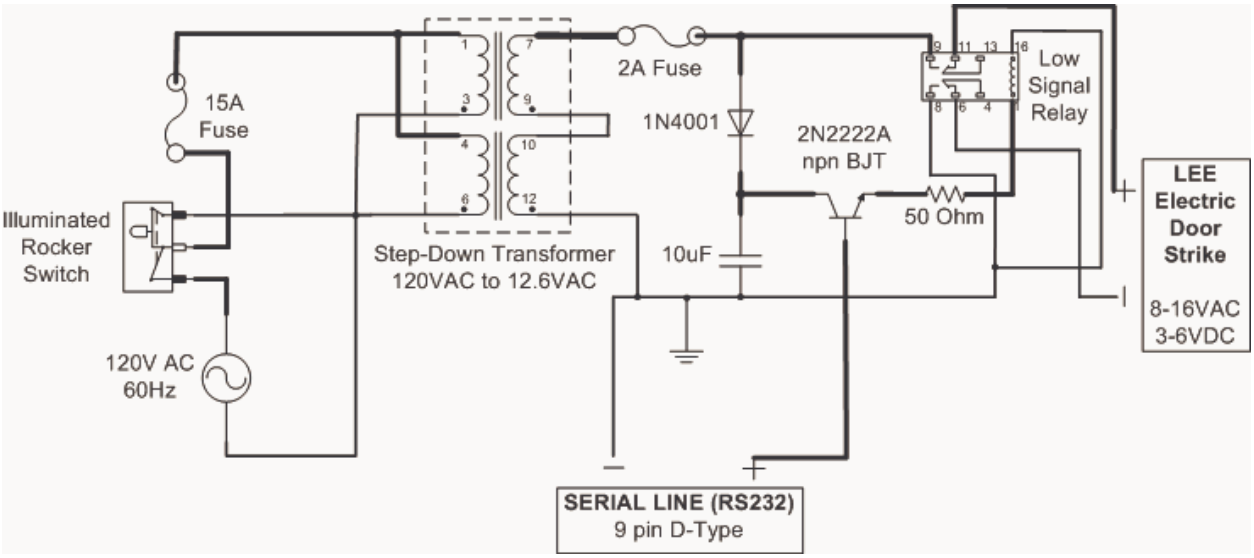


Figure 2: Lock Circuit Schematic

The interface consists of the following components:

1. Power Cord – 120V AC, 60 Hz
2. Illuminated Rocker Switch – This acts as a switch for the entire circuit by breaking the positive voltage line from the power cord. Upon activation, it closes this line and illuminates a red light.
3. Panel Mount Fuse Holder – This holds a fuse rated for 120V, 15A.
4. PC Mount Power Transformer – This is fed by the 120V line from the fuse. With it placed in series connection, the transformer supplies 12.6VAC CT at 2.4A.
5. In-Line Fuse Holder – This holds a fuse rated for 120V, 2A.
6. 1N4001 Diode – This takes a branch of the positive voltage and uses it to power the relay.
7. 10uF Capacitor – This is tied to ground to clean up the BJT's collector voltage.
8. 2N2222 BJT – An npn transistor used as a switch which is triggered by the positive serial line. Its emitter current powers the relay.
9. Low Signal Relay – This acts as a SPST switch for the transformer's output voltage.
10. Serial Port [RS232] – This provides the signal to trigger the BJT. The positive line is pin 3 [SEND] and the negative line is pin 5 [GND]. The connecting cable used is a 9-pin D-Type RS232C.
11. Electric Lock – This can be used in AC or DC applications. For our purposes, it is unlocked by 8-16VAC and draws 1-2A.

Testing

Upon completion of the lock circuit assembly and the m-files necessary for operation, the HUDAT Security System was tested. The tests were performed by recording "master files" for four individuals: 3 males and 1 female. These files were named as the "user name" format utilized in *logon.m* (i.e. "bbash", "dwilliams", etc.). Upon executing *logon*, the user was prompted to enter their user name. This entry was used to reference the original master file. A new user file was created for the session and it was compared to the master file. The numerical results generated by the formant function were used to create the "threshold" (500) utilized in *logon*.

Many tests were executed like this to determine the pass/fail rates. The system's shortcomings can fall into two categories:

1. False negative – This occurs when the user should be authorized, but the system denies access.
2. False positive – This occurs when the user should not be authorized, but the system grants access.

As was predicted using **Table 1**, false negatives occurred more often than false positives. In fact, false positives were fairly rare. Ultimately, this is very good considering it does not compromise the integrity of the system. In the case of our false negatives, they can be attributed to factors such as the room's acoustics and outside disturbances in most instances. Overall, the system performed fairly well.

Cost Analysis

The current lock system in the Biggs Engineering Building of Ohio Northern University is approximately $1000/electronic door lock. This was used as a basis for the cost of the HUDAT Security System.

Table 2 shows the development costs of the software. All of these would impact the HUDAT Company. If the only contract were to replace 100 locks of the current ONU system, the total cost for development and production (100 units) was determined to be $45,000 ($35000 – Development, $10000 – Production).

Table 3 shows the manufacturer's suggested retail prices (MSRP) based upon no profit and 33% profit. As seen from the profit MSRP, the cost is still competitive and favorable when compared to the current $1000/lock system. One issue that was not considered is the number of hours required to have someone reprogram each current lock at least once a year. This would also add to the savings provided by the HUDAT Security System.

Table 2: Estimated Development Costs

Computer System	$800
MATLAB (reusable)	$1,900
MATLAB Signal Processing Toolbox (reusable)	$800
MATLAB Compiler (reusable)	$2,700
Lock & Components	$100
Development ($20/hr, 3 people)	$28,800
TOTAL	$35,100

Table 3: Estimated Product Costs

Break-Even MSRP (per unit)	$450
Profit MSRP (33% profit per unit)	$600

Gantt Chart

The Gantt Chart (**Figure 3**) was created using the program, Microsoft Project, and is a visual layout of the work done throughout the project. With the work being performed on an academic calendar, there are three quarters shown: Fall, Winter, and Spring.

Fall Quarter

Fall Quarter is divided into three main sections: Conceptual, Feasibility, and Proposal.

The Conceptual portion deals with the initial stages of the project. This includes the Team Charter, Problem Identification, and Project Update. It involves any activity that was meant to present the groundwork of the project.

The Feasibility portion deals with discussion of the project. This includes the Block Diagram, Research, Gantt Chart, and Constraint Analysis. It involves any activity that was meant to explain the project and delve more into the subject matter surrounding it.

The Proposal portion simply deals with the final phase of the Fall Quarter. This includes the Rough Draft, Final Draft, and Presentation of the project's proposal.

One area that falls outside of these main sections is the Peer Evaluations. These were conducted to help the team develop and facilitated discussion among the members. The Peer Evaluations will also be repeated numerous times throughout the project.

Winter Quarter

Winter Quarter dealt with a great deal of programming. In fact, programming occurred for the entire duration of the quarter. Research also continued and led to many different directions in the programming.

The Programming portion deals with the actual code writing and the ongoing testing. This testing led to the new direction of implementing formants. Assembly of the lock circuit also occurred. The Update portion deals with the revision of the original Project Proposal.

Spring Quarter

Spring Quarter involved the continuation of programming, as well as testing of the code and the final construction of the lock circuit. Final Documentation involves poster development, revising the written report, developing the oral presentation, and creating the web page.

ID	Task Name	Duration	Start	Finish
1	**Year 1**	**161 days?**	**Mon 9/15/03**	**Mon 5/10/04**
2	**Fall Quarter**	**43 days?**	**Mon 9/15/03**	**Wed 11/12/03**
3	**Conceptual**	**28 days?**	**Mon 9/15/03**	**Wed 10/22/03**
4	Team Charter	5 days?	Mon 9/15/03	Fri 9/19/03
5	Problem Identification	9 days?	Fri 9/19/03	Wed 10/1/03
6	Project Update	16 days?	Wed 10/1/03	Wed 10/22/03
7	**Feasibility**	**40 days?**	**Mon 9/15/03**	**Fri 11/7/03**
8	Block Diagram	9 days?	Fri 9/19/03	Wed 10/1/03
9	Research	40 days?	Mon 9/15/03	Fri 11/7/03
10	Gantt Chart	5 days?	Mon 10/20/03	Fri 10/24/03
11	Constraint Analysis	6 days?	Fri 10/24/03	Fri 10/31/03
12	**Proposal**	**43 days?**	**Mon 9/15/03**	**Wed 11/12/03**
13	Proposal - Rough draft	38 days?	Mon 9/15/03	Wed 11/5/03
14	Proposal - Final	40 days?	Mon 9/15/03	Fri 11/7/03
15	Proposal - Presentation	5 days?	Thu 11/6/03	Wed 11/12/03
16	Peer Evaluations	25 days?	Mon 9/15/03	Fri 10/17/03
17	**Winter Quarter**	**55 days?**	**Mon 12/1/03**	**Fri 2/27/04**
18	**Implementation**	**55 days?**	**Mon 12/1/03**	**Fri 2/27/04**
19	Programming [Coding & Testing]	53 days?	Mon 12/1/03	Wed 2/25/04
20	Assembly	16 days?	Thu 2/5/04	Thu 2/26/04
21	Testing	2 days?	Thu 2/26/04	Fri 2/27/04
22	Peer Evaluations	55 days?	Mon 12/1/03	Fri 2/27/04
23	Update	4 days?	Tue 2/24/04	Fri 2/27/04
24	**Spring Quarter**	**46 days?**	**Mon 3/8/04**	**Mon 5/10/04**
25	Implementation	15 days?	Mon 3/8/04	Fri 3/26/04
26	Testing	15 days?	Mon 3/29/04	Fri 4/16/04
27	Final Assembly	1 day?	Wed 4/28/04	Wed 4/28/04
28	**Final Documentation**	**16 days?**	**Mon 4/19/04**	**Mon 5/10/04**
29	Poster Development	4 days?	Mon 4/19/04	Thu 4/22/04
30	Written Report	10 days?	Mon 4/19/04	Fri 4/30/04
31	Oral Presentation	4 days?	Fri 4/30/04	Wed 5/5/04
32	Web Page	6 days?	Mon 5/3/04	Mon 5/10/04

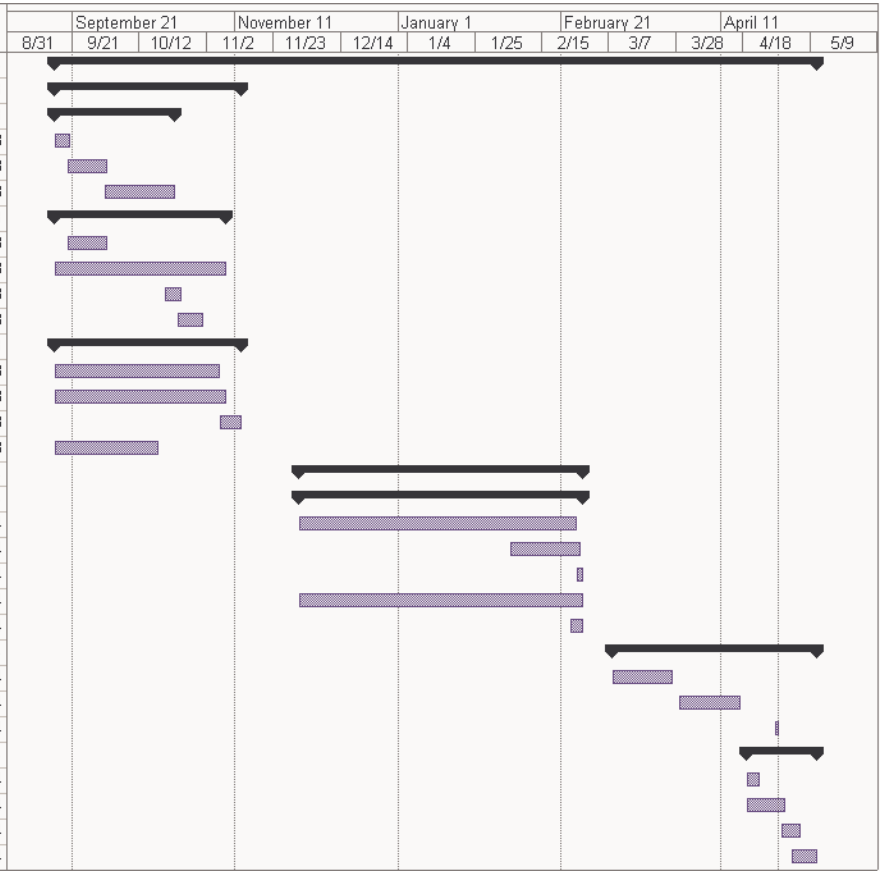


Figure 3: Project Gantt Chart

Conclusion

A speaker verification system in a security application was developed. It utilized voice biometrics to distinguish between authorized users. The system was comprised of MATLAB code, a circuit to activate a lock, and the electric door lock itself. Numerous tests were conducted to determine its effectiveness. The findings of the tests revealed that the results were within the expected performance parameters. Also, a cost analysis was performed to compare it to a comparable security system already in use.

References

[1] "Biometrics Comparison Chart." Court Technology Laboratory. Retrieved 07 Nov. 2003 <http://ct.ncsc.dni.us/biomet%20web/BMCompare>.

[2] Markowitz, Judith A. "Voice Biometrics". Communications of the ACM. Vol. 43, No. 9. September 2000. p66-73.

[3] Ellis, E. Darren. "Design of a Speaker Recognition Code using MATLAB." *16-199 Assignment 1*. 27 January 2004. http://www.andrew.cmu.edu/~asurie/199-1.htm

Appendix A – M-file Code

deriv.m

```
% deriv.m - Thomas H. Jonell
% create the derivative of a function.
% returns the value of the max, and its index, along
% with the derivative of the function.
% This function generally replaces the previously
% used peak finder functions.

function [yq,max,ind] = deriv(xq)

% set up globals
slope = 0;
pslope = 0;
c = 0;  % counter for max positioning

for i = 1:1:(length(xq)-1)
    % calculate slope and store it in output variable
    slope = (xq(i+1)-xq(i))/((i+1)-i);
    yq(i) = slope;
    % if there is a maximum detected, record its position and value.
    if (pslope > 0) & (slope < 0)
        c = c + 1;
        ind(c) = i;
        max(c) = xq(i);
    end
    % store past slope value
    pslope = slope;
end
```

formant.m

```
% formant.m - Thomas H. Jonell
% creates a vector containing the formants in a sound file

function [apx,max,ind] = formant(file)

% read in wav file
[xq,fs,nb] = wavread([file]);

% perform the yule-walker spectral power density calculations
pxq = pyulear(xq,12);
apx = 20*log10(pxq);

% find the peaks of the formants
[yq,max,ind] = deriv(apx);

newtest.m
% newtest.m Thomas H. Jonell
% 2-24-04
% test the given formant indexes and return
% their difference factor

function passfail = newtest(ind1,ind2)

% get the initial sizes of the indices
L1 = length(ind1);
L2 = length(ind2);
```

```
% find the smallest if they are different sizes
if L1 < L2
    L = L1;
else
    L = L2;
end
c = 0;
% "remove" all data below index 20 for ind1
for i = 1:L
    if ind1(i) < 20
        c = i;
    end

end
% create new index1
i1 = ind1((c+1):L1);


% "remove" all data below index 20 for ind2
for i = 1:L
    if ind2(i) < 20
        c = i;
    end
end
% create new index2
i2 = ind2((c+1):L2);

% get the lengths of the new indices
L1 = length(i1);
L2 = length(i2);

% if the indices are not the same size, immediately fail the test
if L1 ~= L2
    passfail = -1;
    return;
end

% since the indices are the same size, test them.
z = 0;
for i = 1:L1    % which length doesnt matter because they are the same size
    x = i1(i)-i2(i);
    y = x * x;
    z = y + z;
end
passfail = z;
```

lock.m

```
% lock.m - Thomas Jonell & Brian Bash
% 4-21-04
% sends data to serial port to open lock

a = int32(0);
s = serial('COM1','BaudRate',9600,'Parity','none');
fopen(s);

% ~5 sec of open time
for i = 1:1:2500
    fwrite(s,a);
end
```

```
fclose(s);
instrfind;

delete(s);
clear s;
clear a;
clear ans;
clear i;
```

logon.m

```
% HUDAT SECURITY SYSTEM
% HUDAT Members: Brian Bash, Tom Jonell, Dustin Williams
% HUDAT Advisor: Dr. Les Thede
% Senior Project 2003-04

% clear display
clc
clear all
format compact
close all

while (1==1)
    % loop forever
    while (1 == 1)
        count = 0;
        threshold = 500;

        % initialize display
        disp(' ')
        disp('        ******************************')
        disp('        *                            *')
        disp('        *       Welcome to the       *')
        disp('        *    Hudat Security System   *')
        disp('        *                            *')
        disp('        ******************************')
        disp(' ')
        disp(' ')

        % user logon
        name = input('Please begin by entering your user name: ','s');
        % get the respective wav file
        file = [name,'.wav'];
        disp(' ')

        % See if the person is allowed in the room
        if ((exist (file))<1)
            disp('The user name entered is not valid.  Press any key to reset the
              system.')
            pause
            clc
            clear all
            break;
        end

        % initialize values
        pass = 0;
        rec = 1;
```

```
% Recording/testing loop
while(rec == 1)
    % wav file cleanup
    if (exist('user.wav'))
        delete('user.wav');
    end
    if (exist('a1.wav'))
        delete('a1.wav');
    end
    if (exist('a2.wav'))
        delete('a2.wav');
    end
    if (exist('a3.wav'))
        delete('a3.wav');
    end
    if (exist('a4.wav'))
        delete('a4.wav');
    end
    if (exist('b1.wav'))
        delete('b1.wav');
    end
    if (exist('b2.wav'))
        delete('b2.wav');
    end
    if (exist('b3.wav'))
        delete('b3.wav');
    end
    if (exist('b4.wav'))
        delete('b4.wav');
    end

    disp('When you are ready to begin, press any key and recite')
    disp('the passphrase slowly into the microphone.')
    disp(' ')
    pause

    % record new file
    recwav('user')
    disp('PROCESSING...')
    disp(' ')

    % create file prefix string
    a = 'a';
    b = 'b';
    user = 'user.wav';

    % split the wav files
    newsplit([user],[a]);

    % Make sure it recorded OK
    if ((exist('a1.wav') == 0) | (exist('a2.wav') == 0) | (exist('a3.wav') == 0) |
        (exist('a4.wav') == 0))
        disp('The system could not recognize the passphrase.  Press any key to
             retry.')
        disp(' ')
        pause
    else
        newsplit([file],[b]);

        % perform formant (comparison) analysis & test of indexes

        % "Please"
        [ap1,m1,i1] = formant('a1.wav');
```

```
[ap2,m2,i2] = formant('b1.wav');
pass1 = newtest(i1,i2);

% "Let"
[ap1,m1,i1] = formant('a2.wav');
[ap2,m2,i2] = formant('b2.wav');
pass2 = newtest(i1,i2);

% "Me"
[ap1,m1,i1] = formant('a3.wav');
[ap2,m2,i2] = formant('b3.wav');
pass3 = newtest(i1,i2);

% "In"
[ap1,m1,i1] = formant('a4.wav');
[ap2,m2,i2] = formant('b4.wav');
pass4 = newtest(i1,i2);

% Determining score
% Adjustment for poor performance --> Variable threshold+
% pass1
if ((pass1 < 0) | (pass1  > threshold))
    pass1 = -1;
elseif (pass1 <= 25)
    pass1 = 2;
elseif ((pass1 > 25) & (pass1 <=250))
    pass1 = 1;
else
    pass1 = 0;
end

% pass2
if ((pass2 < 0) | (pass2  > threshold))
    pass2 = -1;
elseif (pass1 <= 25)
    pass2 = 2;
elseif ((pass1 > 25) & (pass1 <=250))
    pass2 = 1;
else
    pass2 = 0;
end

% pass3

if ((pass3 < 0) | (pass3  > threshold))
    pass3 = -1;
elseif (pass3 <= 25)
    pass3 = 2;
elseif ((pass3 > 25) & (pass3 <=250))
    pass3 = 1;
else
    pass3 = 0;
end

%pass4
if ((pass4 < 0) | (pass4  > threshold))
    pass4 = -1;
elseif (pass4 <= 25)
    pass4 = 2;
elseif ((pass4 > 25) & (pass4 <=250))
    pass4 = 1;
else
    pass4 = 0;
```

```
                end

                % Determine access/no access
                pass = pass1 + pass2 + pass3 + pass4;

                if ((pass <= 0) & (count < 2))
                    disp('This is not a valid match.')
                    disp(' ')
                    count = count + 1;
                elseif ((pass <= 0) & (count >= 2))
                    disp('This is not a valid match.  Attempt limit reached.  Press any
                         key to reset.')
                    disp(' ')
                    pause

                    %reset
                    clc
                    clear all
                    break;
                else
                    disp('Access granted.  Welcome.')
                    %unlock door
                    lock;
                    %reset
                    clc
                    clear all
                    break;
                end
            end
        end
    end
end
```

newsplit.m

```
% newsplit.m - Thomas H. Jonell
% new splitter program, much, much faster than the old split.m

function newsplit(wavfile,extension)

% read in wavfile
[xq,Fs,B] = wavread([wavfile]);

% create postfix and prefix strings
postf = '.wav';
pref = extension;

% create the pitch data
x = pitch(xq);

% find lengths of both files
Lx = length(x);
Lxq = length(xq);

% find the ratio of the 2 data files for translation
r = ceil(Lxq/Lx);

% calculate the mean for use as the threshold
m = mean(x);

% create dummy index values
```

```
p = 1;
n = r;

% create flag values for spooken start and finish
ss = 0;
fs = 0;

% create a counter for wav file labeling
counter = 0;

% test pitch file to find spoken sections
for i = 1:Lx
    % start of a spoken portion
    if (x(i) > m) & (ss == 0)
        % set start index and start flag
        start = i - 13;
        ss = 1;
    elseif (x(i) < m) & (ss == 1) & (fs == 0) % end of a spoken portion
        % set finish index and finish flag
        finish = i + 13;
        fs = 1;
    end
    % there was a spoken portion, write the file
    if (ss == 1) & (fs == 1)
        % translate indexes
        p = start * r;
        n = finish * r;
        % make sure its not larger than the original file index
        if n > Lxq
            n = Lxq;
        end

        % test to make sure the "spoken" segment is large enough
        q = n - p;
        if q > 2000
            % copy values to temporary variable
            z = xq(p:n);
            % update counter and create a string of it
            counter = counter + 1;
            num = mat2str(counter);
            % write the file
            wavwrite(z,Fs,B,[pref,num,postf]);
        end
        % reset flags
        ss = 0;
        fs = 0;

    end
end
```

newtest.m

```
% newtest.m Thomas H. Jonell
% test the given formant indexes and return
% their difference factor

function passfail = newtest(ind1,ind2)

% get the initial sizes of the indices
L1 = length(ind1);
L2 = length(ind2);
```

```
% find the smallest if they are different sizes
if L1 < L2
    L = L1;
else
    L = L2;
end
c = 0;
% "remove" all data below index 20 for ind1
for i = 1:L
    if ind1(i) < 20
        c = i;
    end

end
% create new index1
i1 = ind1((c+1):L1);


% "remove" all data below index 20 for ind2
for i = 1:L
    if ind2(i) < 20
        c = i;
    end
end
% create new index2
i2 = ind2((c+1):L2);

% get the lengths of the new indices
L1 = length(i1);
L2 = length(i2);

% if the indices are not the same size, immediately fail the test
if L1 ~= L2
    passfail = -1;
    return;
end

% since the indices are the same size, test them.
z = 0;
for i = 1:L1    % which length doesnt matter because they are the same size
    x = i1(i)-i2(i);
    y = x * x;
    z = y + z;
end
passfail = z;
```

pitch.m

```
% pitch.m - Thomas H. Jonell
% returns a vector of autocorrelated maxes of a given vector
% used to determine spoken portions of a sound file.

function pit = pitch(xq)

% find the length of the sample
L = length(xq);

% create the window
p = 1;
n = 100;
```

```
% find how large half the size of the window is
d = ceil((n - p)/2);
% find the size of pit.
L2 = floor((L / d));

% perform the autocorrelation of the windows and create
% the pitch file.
for i = 1:1:L2
    if n > L
        n = L;
    end
    b = xcorr(xq(p:n));
    % find the max of the autocorrelated window and store
    % the value in pit.
    v = max(b);
    pit(i,1) = v(1);
    % update window position
    p = p + 50;
    n = n + 50;
end
```

recwav.m

```
% recwav.m - Thomas H. Jonell
% records a wavfile

function recwav(name)

extType = '.wav';
Fs = 11025;
N = 16;
CH = 1;
filename = [[name],[extType]];

y = wavrecord(5*Fs,Fs,'double');
wavwrite(y,Fs,N,[filename]);

return;
```

Appendix B – Sample User Interfaces

Example 1: Successful Login Attempt

```
        ******************************
        *                            *
        *       Welcome to the       *
        *    Hudat Security System   *
        *                            *
        ******************************


Please begin by entering your user name: bbash

When you are ready to begin, press any key and recite
the passphrase slowly into the microphone.

PROCESSING...

Access granted.  Welcome.
```

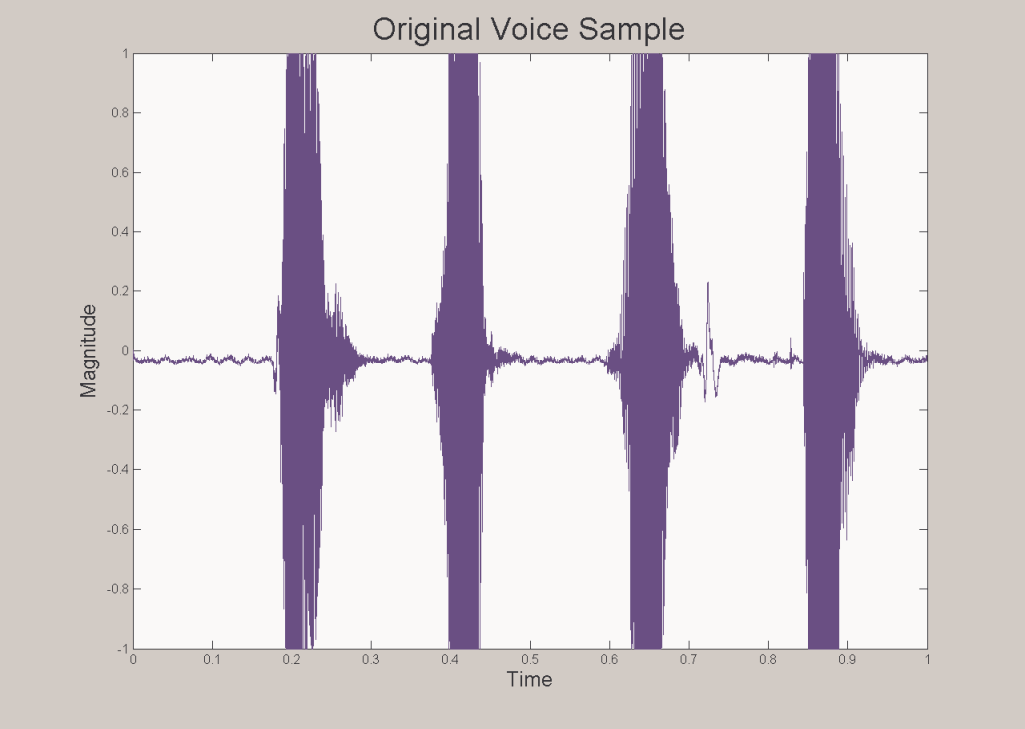
Example 2: Three Failed Login Attempts

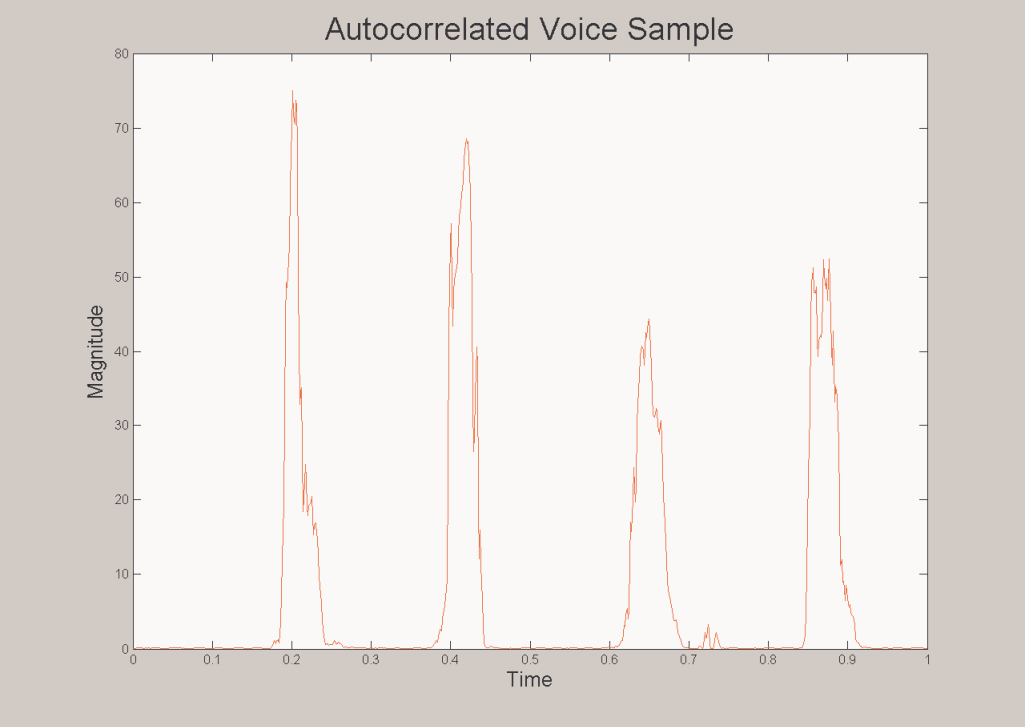
```
        ******************************
        *                            *
        *       Welcome to the       *
        *    Hudat Security System   *
        *                            *
        ******************************


Please begin by entering your user name: swagner

When you are ready to begin, press any key and recite
the passphrase slowly into the microphone.

PROCESSING...

This is not a valid match.

When you are ready to begin, press any key and recite
the passphrase slowly into the microphone.

PROCESSING...

This is not a valid match.

When you are ready to begin, press any key and recite
the passphrase slowly into the microphone.

PROCESSING...

This is not a valid match.  Attempt limit reached.  Press any key to reset.
```

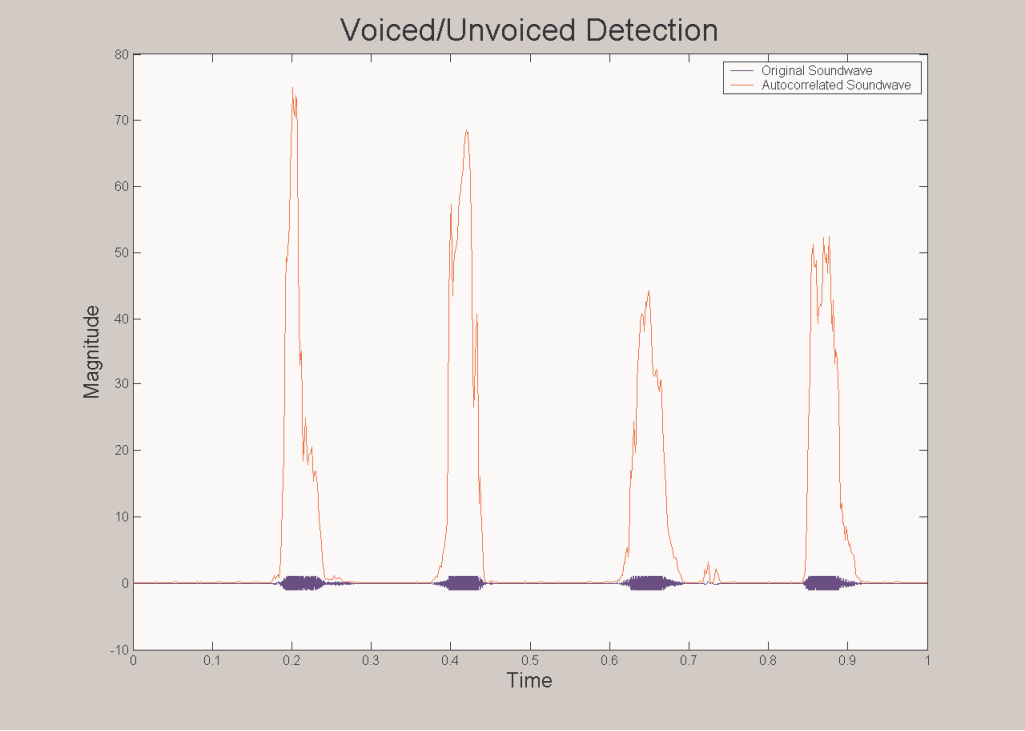
Example 3: Invalid User Name

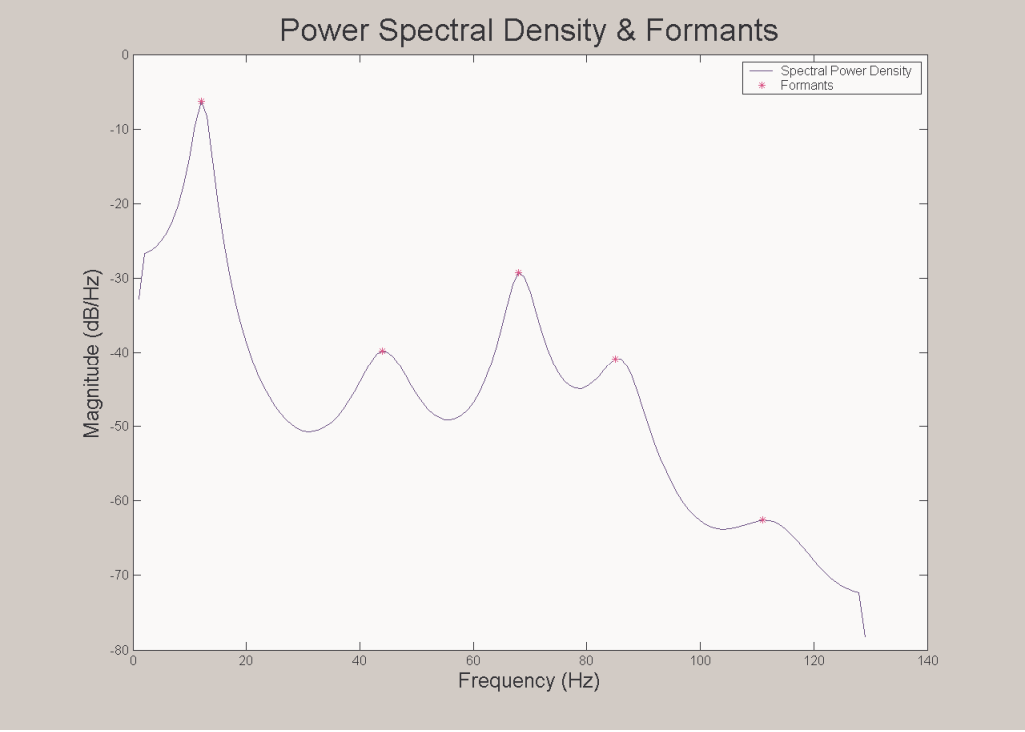
```
*******************************
*                             *
*       Welcome to the        *
*    Hudat Security System    *
*                             *
*******************************

Please begin by entering your user name: lthede

The user name entered is not valid.  Press any key to reset the system.
```

Appendix C – MATLAB Plots






Voiced/Unvoiced Detection
Original Soundwave
Autocorrelated Soundwave
Magnitude
Time
80
70
60
50
40
30
20
10
0
-10
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1


Power Spectral Density & Formants
Spectral Power Density
Formants
Magnitude (dB/Hz)
Frequency (Hz)
0
-10
-20
-30
-40
-50
-60
-70
-80
0
20
40
60
80
100
120
140